Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
OPENMP IMPLEMENTATION
KIN TAT, TAN

OPENMP IMPLEMENTATION

KIN TAT, TAN

Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
OPENMP IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

Summary

Universitat de Lleida

List of images

Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
OPENMP IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

GitHub

https://github.com/tankintat/highperformanceudl

Description problem

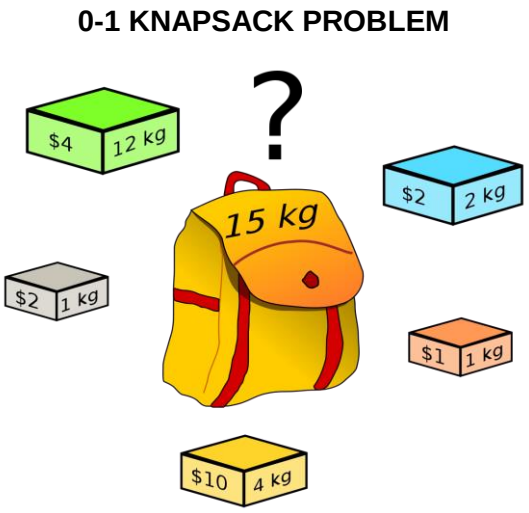

Figura 1 - Knapsack problem

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

Universitat de Lleida

Description solution

The problem has two ways to resolve, Dynamic Programming approach or Branch and Bounce approach to organize the idea to paralelize the algoritm in order to increase the speed of execution. The serial algorithm execute runs linearly that means try to found the better way searching all ways to find the maximum weight with the better value.

The serial solution has 2 loops, one to control the quantity of inputs, and the inside control each weight increasing one by one up to the maximum weight. This makes it possible to test all combinations.

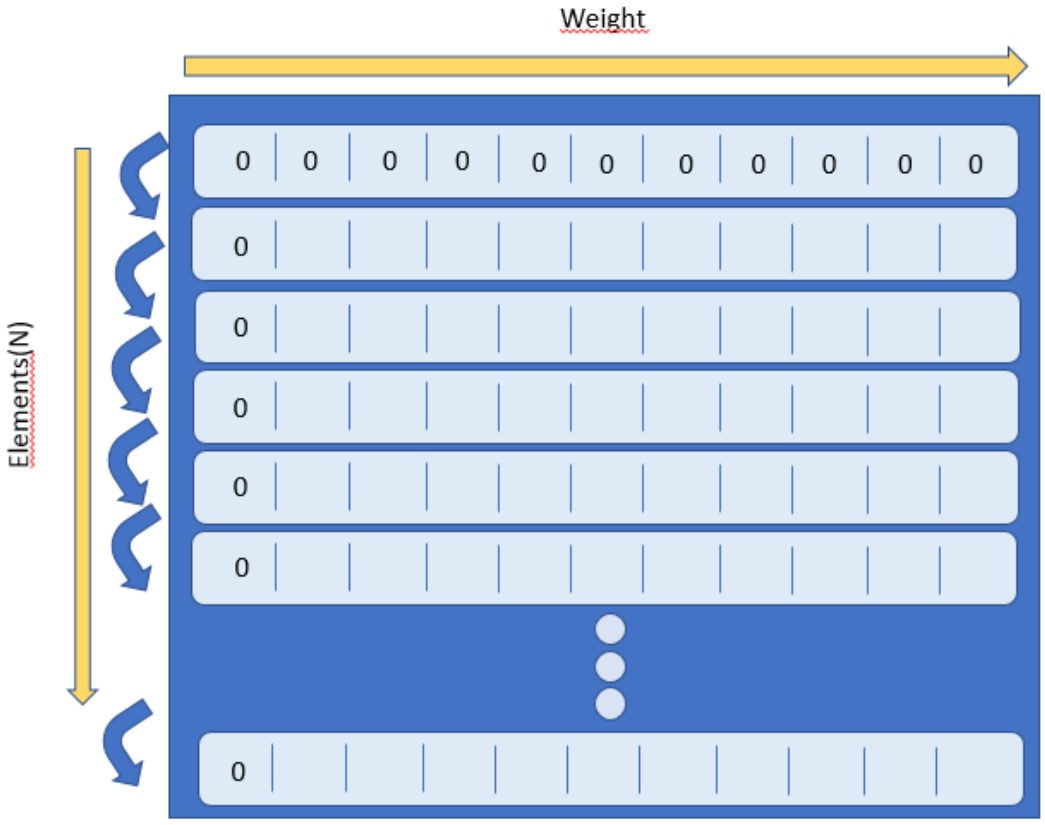

Figura 2 - Knapsack serial problem

On the inside loop to execute in parallel all weights the same time and the barrier to wait and restart to next one again. We can put on the external loop because on the algorithm need the last row when overcome the sack weight in this case its necessary to get the last sum weight.

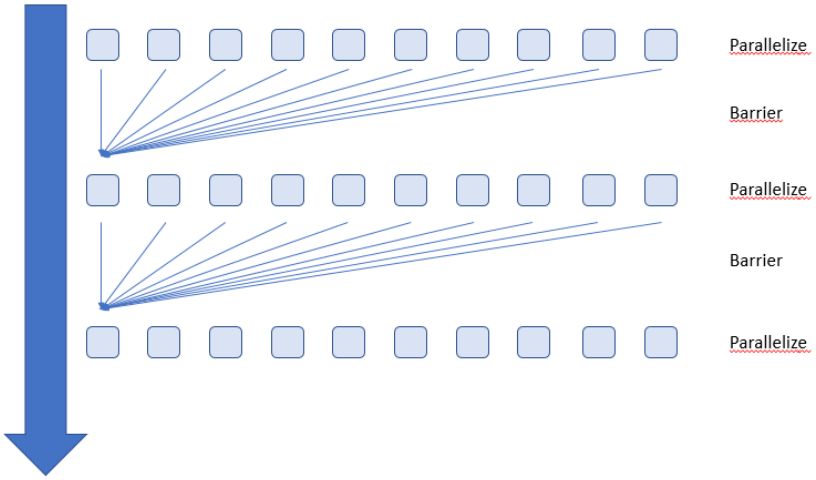

Figura 3 - Knapsack parallel problem.

Development

For the development it was necessary the server moore of the University used to the tests. The first was connected of the moore server and put the folder base (hpc-project) to start.

Figura 4 - Modifications on the code.

Using the qlogin command to execute the serial and parallel programs.

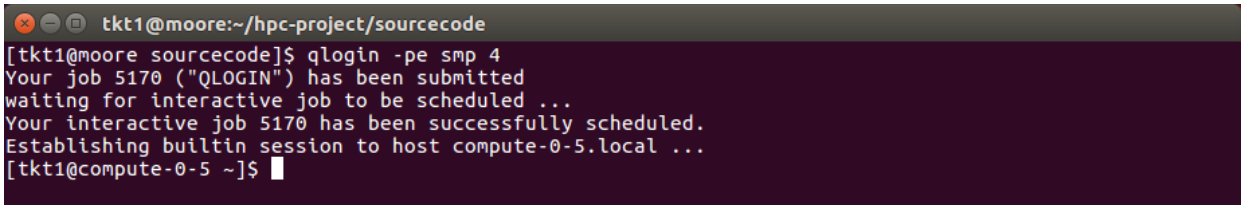

Figura 5 - Using the qlogin command to connect with the one computer.

And the end execute a file batch to execute all programs serials and parallels to compare the outputs. Compile the file before to execute the running_results.sh.

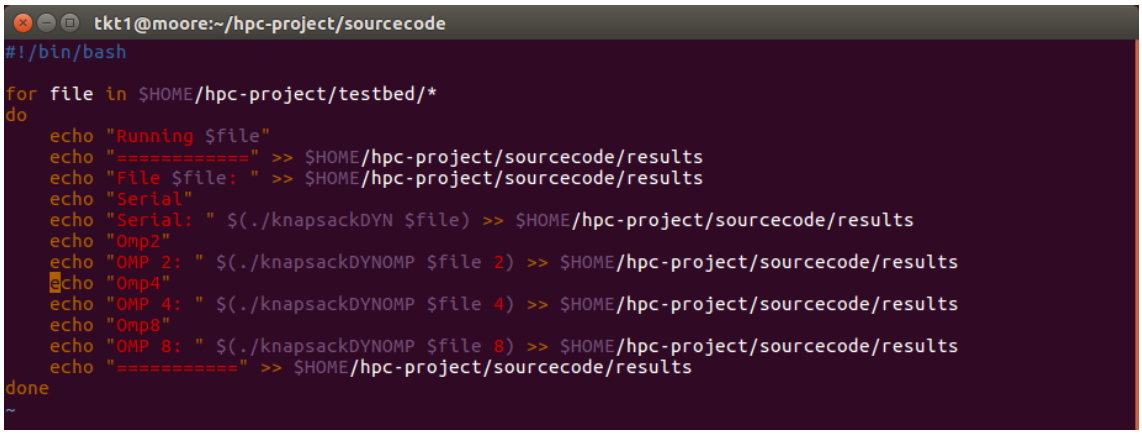

Figura 6 - File batch to execute all files (Serial/Parallel (Thread2/4/8).

Universitat de Lleida

Results

The complete results you can see openning this link:

https://github.com/tankintat/highperformanceudl/blob/master/OpenMP_Implementations/results or the file "result" package together with this file.

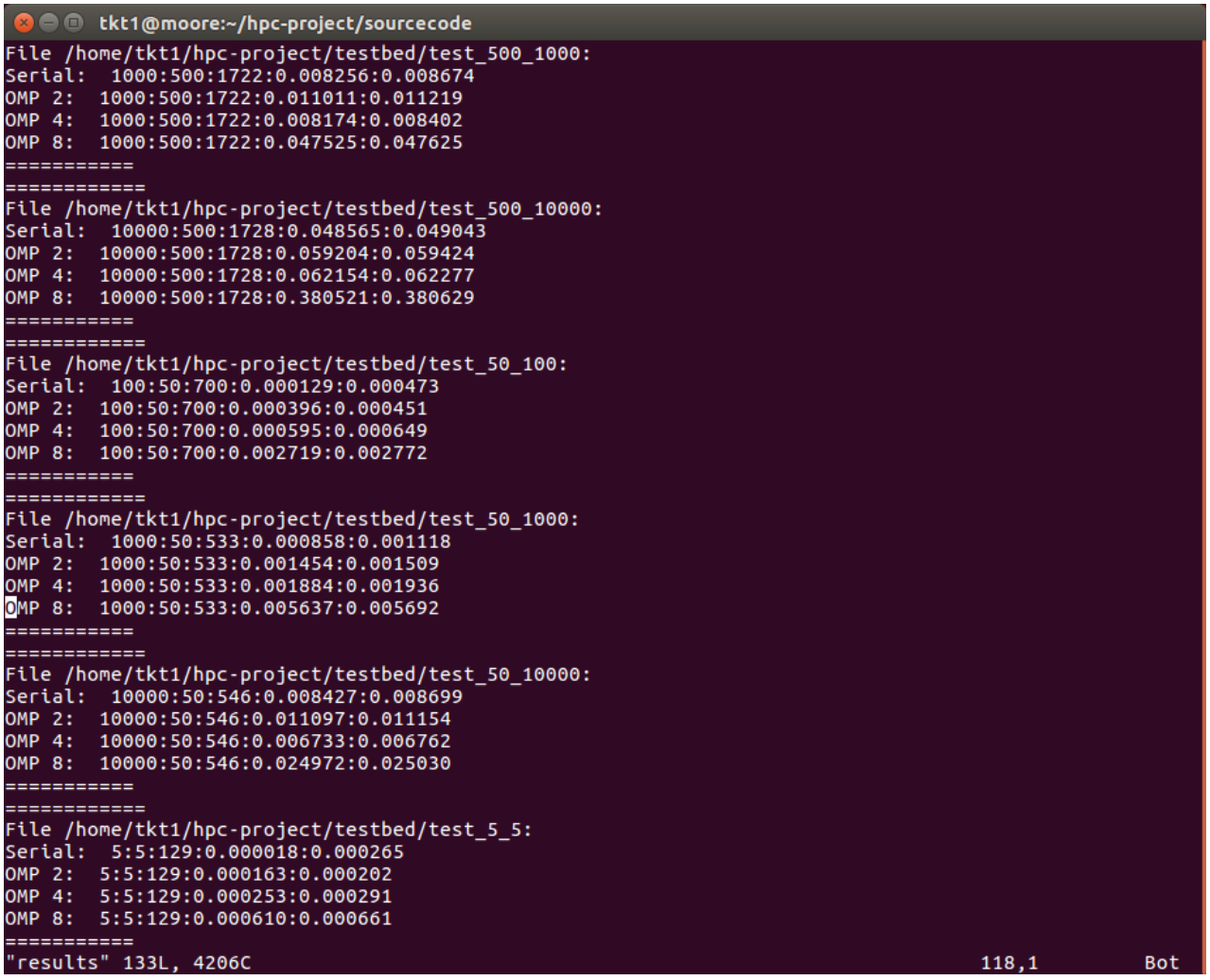

Figura 7 - Results

Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
OPENMP IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

Conclusions

We can observe that have three files that make failure with error of the threads and execution, we also can obsever all the executions with 8 threads take longer time;

I conclude with the smaller inputs not have much difference in the serial and the parallel with 2 and 4 threads. A case "test_50000_1000" for serial was generated 3.9 sec and for the parallel 6.0 and 6.4, i suppose that some bottleneck in the execution, because the another result with the same quantity increasing the value of weight and results giving similar.

The another way we can test its transform the used matrix to vector because the indexing of vector its more faster, but the address the matrix in the memory its a similar of vector.

This graph we can observe that the unique that have a difference when the quantity of input increase its the 8 threads.

				Serial	OMP2	OMP4	OMP8
	5	5		0,000265	0,000202	0,000291	0,000661
	50	100		0,000473	0,000451	0,000649	0,002772
	50	1000		0,001118	0,001509	0,001936	0,005692
	50	10000		0,008699	0,011154	0,006762	0,02503
	500	100		0,017001	0,002791	0,003903	0,025939
	500	1000		0,008674	0,011219	0,008402	0,047625
	500	10000		0,049043	0,059424	0,062277	0,380629
	5000	100		0,025165	0,018768	0,0178	0,256032
	5000	1000		0,049805	0,067701	0,076332	0,598548
	5000	10000		0,418807	0,592808	0,631391	4
	50000	100		0,064808	0,12621	0,169211	2
	50000	1000		0,427239	0,664573	0,748265	6
	50000	10000		4	6	6	39
	500000	100		1	1	2	26
	500000	1000		4	7	7	60
	500000	10000					
	5000000	100		50	13	17	
	5000000	1000					
	5000000	10000